Chapter 1

INTRODUCTION

Corruption of data is a fact of life. Error-correcting codes (or just codes) are clever ways of representing data so that one can recover the original information even if parts of it are corrupted. The basic idea is to judiciously introduce redundancy so that the original information can be recovered even when parts of the (redundant) data have been corrupted.

Perhaps the most natural and common application of error correcting codes is for communication. For example, when packets are transmitted over the Internet, some of the packets get corrupted or dropped. To deal with this, multiple layers of the TCP/IP stack use a form of error correction called CRC Checksum [87]. Codes are used when transmitting data over the telephone line or via cell phones. They are also used in deep space communication and in satellite broadcast (for example, TV signals are transmitted via satellite). Codes also have applications in areas not directly related to communication. For example, codes are used heavily in data storage. CDs and DVDs work fine even in presence of scratches precisely because they use codes. Codes are used in Redundant Array of Inexpensive Disks (RAID) [24] and error correcting memory [23]. Codes are also deployed in other applications such as paper bar codes, for example, the bar code used by UPS called MaxiCode [22].

In this thesis, we will think of codes in the communication scenario. In this framework, there is a sender who wants to send (say) k message symbols over a noisy channel. The sender first *encodes* the k message symbols into n symbols (called a *codeword*) and then sends it over the *channel*. The receiver gets a *received word* consisting of n symbols. The receiver then tries to *decode* and recover the original k message symbols. The main challenge in coding theory is to come up with "good" codes along with efficient encoding and decoding algorithms. In the next section, we will define more precisely the notion of codes and the noise model.

Typically, the definition of a code gives the encoding algorithm "for free." The decoding procedure is generally the more challenging algorithmic task. In this thesis, we concentrate more on the decoding aspect of the problem. In particular, we will consider two relaxations of the "usual" decoding problem in which either the algorithm outputs the original message that was sent or gives up (when too many errors have occurred). The two relaxations are called *list decoding* and *property testing*. The motivations for considering these two notions of decoding are different: list decoding is motivated by a well known limit on the number of errors one can decode from using the usual notion of decoding while property

testing is motivated by a notion of "spot-checking" of received words that has applications in complexity theory. Before we delve into more details of these notions, let us first review the basic definitions that we will need.

1.1 Basics of Error Correcting Codes

We will now discuss some of the basic notions of error correcting codes that are needed to put forth the contributions of this thesis.[1] These are the following.

- Encoding The *encoding function* with parameters k, n is a function $E : \Sigma^k \to \Sigma^n$, where Σ is called the *alphabet*. The encoding function E takes a *message* $m \in \Sigma^k$ and converts it into a *codeword* $E(m)$. We will refer to the algorithm that implements the encoding function as an *encoder*.

- Error Correcting Code An *error correcting code* or just a *code* corresponding to an encoding function E is just the image of the encoding function. In other words, it is the collection of all the codewords. A code C with encoding function $E : \Sigma^k \to \Sigma^n$ is said to have *dimension* k and *block length* n. In this thesis, we will focus on codes of large block length.

- Rate The ratio $R = k/n$ is called the *rate* of a code. This notion captures the amount of redundancy used in the code. This is an important parameter of a code which will be used throughout this thesis.

- Decoding Consider the basic setup for communication. A *sender* has a message that it sends as a codeword after encoding. During transmission the codeword gets distorted due to errors. The *receiver* gets a noisy *received word* from which it has to recover the original message. This "reverse" process of encoding is achieved via a *decoding function* $D : \Sigma^n \to \Sigma^k$. That is, given a received word, the decoding function picks a message that it thinks was the message that was sent. We will refer to the algorithm that implements the decoding function as a *decoder*.

- Distance The *minimum distance* (or just *distance*) of a code is a parameter that captures how much two different codewords differ. More formally, the distance between any two codewords is the number of coordinates in which they differ. The (minimum) distance of a code is the minimum distance between any two distinct codewords in the code.

---

[1] We will define some more "advanced" notions later.

1.1.1 Historical Background and Modeling the Channel Noise

The notions of encoding, decoding and the rate appeared in the seminal work of Shannon [94]. The notions of codes and the minimum distance were put forth by Hamming [67].

Shannon modeled the noise *probabilistically*. For such a channel, he also defined a real number called the *capacity*, which is an upper bound on the rate of a code for which one can have reliable communication. Shannon also proved the converse result. That is, there *exist* codes for any rate less than the capacity of the channel for which one can have reliable communication. This striking result essentially kick-started the fields of information theory and coding theory.

Perhaps an undesirable aspect of Shannon's noise model is that its effectiveness depends on how well the noise is modeled. In some cases it might not be possible to accurately model the channel. In such a scenario, one option is to model the noise *adversarialy*. This was proposed by Hamming. In Hamming's noise model, we think of the channel as an adversary who has the full freedom in picking the location as well as nature of errors to be introduced. The only restriction is on the number of errors. We will consider this noise model in the thesis.

Alphabet Size and the Noise Model

We would like to point out that the noise model is intimately tied with the alphabet. A symbol in the alphabet is the "atomic" unit on which the noise acts. In other words, a symbol that is fully corrupted and a symbol that is partially corrupted are treated as the same. That is, the smaller the size of the alphabet, the more fine-grained the noise. This implies that the decoder has to take care of more error patterns for a code defined over a smaller alphabet. As a concrete example, say we want to design a decoder that can handle 50% of errors. Consider a code C that is defined over an alphabet of size 4 (i.e., each symbols consists of two bits). Now, let e be an error pattern in which every alternate bit of a codeword in C is flipped. Note that this implies that *all* the symbols of the codeword have been corrupted and hence the decoder does not need to recover from e. However, if C were defined over the binary alphabet then the decoder would have to recover from e. Thus, it is harder to design decoders for codes over smaller alphabets.

Further, the noise introduced by the channel should be independent of the message length. However, in this thesis, we will study codes that are defined over alphabets whose size depends on the message length. In particular, the number of bits required to represent any symbol in the alphabet would be logarithmic in the message length. The reason for this is two-fold: As was discussed in the paragraph above, designing decoding algorithms is strictly easier for codes over larger alphabets. Secondly, we will use such codes as a starting point to design codes over fixed sized alphabets.

With the basic definitions in place, we now turn our attention to the two relaxations of the decoding procedure that will be the focus of this thesis.

1.2 List Decoding

Let us look at the decoding procedure in more detail. Upon getting the noisy received word, the decoder has to output a message (or equivalently a codeword) that it thinks was actually transmitted. If the output message is different from the message that was actually transmitted then we say that a decoding error has taken place. For the first part of the thesis, we will consider decoders that do not make any decoding error. Instead, we will consider the following notion called *unique decoding*. For any received word, a unique decoder either outputs the message that was transmitted by the sender or reports a decoding failure.

One natural question to ask is how many errors such a unique decoder can tolerate. That is, is there a bound on the number of errors (say $\rho_U n$, so ρ_U is the fraction of errors) such that for any error pattern with total error at most $\rho_U n$, the decoder always outputs the transmitted codeword?

We first argue that $\rho_U \leqslant 1 - R$. Note that the codeword of n symbols really contains k symbols of information. Thus, the receiver should have at least k uncorrupted symbols among the n symbols in the received word to have any hope of recovering the transmitted message. In other words, the information theoretic limit on the number of errors from which one can recover is $n - k$. This implies that $\rho_U \leqslant (n-k)/n = 1 - R$. Can this information theoretic limit be achieved ?

Before answering the question above, we argue that the limit also satisfies $\rho_U < d/(2n)$, where we assume that the distance of the code d is even. Consider two distinct messages m_1, m_2 such that the distance between $E(m_1)$ and $E(m_2)$ is exactly d. Now say that the sender sends the codeword $E(m_1)$ over the channel and the channel introduces $d/2$ errors and distorts the codeword into a received word $\mathbf{y}$ that is at a distance of $d/2$ from *both* $E(m_1)$ and $E(m_2)$ (see Figure 1.1).

Now, when the decoder gets $\mathbf{y}$ as an input it has no way of knowing whether the original transmitted codeword was $E(m_1)$ or $E(m_2)$.[2] Thus, the decoder has to output a decoding failure when it receives $\mathbf{y}$ and so we have $\rho_U < d/(2n)$. How far is $d/(2n)$ from the information theoretic bound of $1 - R$? Unfortunately the gap is quite big. By the so called Singleton bound, $d \leqslant n - k + 1$ or $d/n < 1 - R$. Thus, the limit of $d/(2n)$ is at most half the information theoretic bound. We note that even though the limits differ by "only a small constant," in practice the potential to correct twice the number of errors is a big gain.

Before we delve further into this gap between the information theoretic limit and half the distance bound, we next argue that the the bound of $d/2$ is in fact tight in the following sense. If $\rho_U n = d/2 - 1$, then for an error pattern with at most $\rho_U n$ errors, there is always a unique transmitted codeword. Suppose that this were not true and let $E(m_1)$ be the transmitted codeword and let $\mathbf{y}$ be the received word such that $\mathbf{y}$ is within distance $\rho_U n$ from both $E(m_1)$ and $E(m_2)$. Then by the triangle inequality, the distance between $E(m_1)$ and $E(m_2)$ is at most $2\rho_U n = d - 2 < d$, which contradicts the fact that d is the

[2]Throughout this thesis, we will be assuming that the only communication between the sender and the receiver is through the channel and that they do not share any side information/channel.

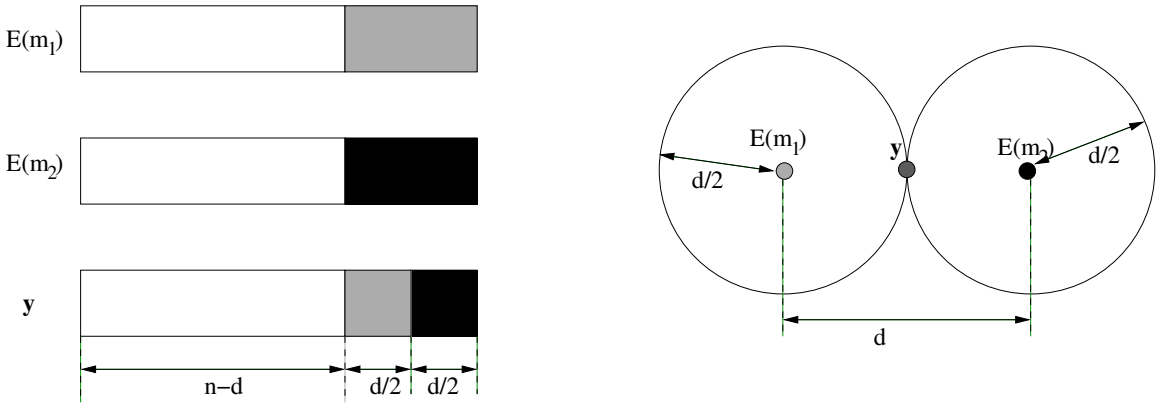

Figure 1.1: Bad example for unique decoding. The picture on the left shows two codewords $E(m_1)$ and $E(m_2)$ that differ in exactly d positions while the received word $\mathbf{y}$ differs from both $E(m_1)$ and $E(m_2)$ in $d/2$ many positions. The picture on the right is another view of the same example. Every n-symbol vector is now drawn on the plane and the distance between any two points is the number of positions they differ in. Thus, $E(m_1)$ and $E(m_2)$ are at a distance d and $\mathbf{y}$ is at a distance $d/2$ from both. Further, note that any point that is strictly contained within one of the balls of radius $d/2$ has a unique closest-by codeword.

minimum distance of the code (also see Figure 1.1). Thus, as long as $\rho_U n = d/2 - 1$, the decoder can output the transmitted codeword. So if one wants to do unique decoding then one can correct up to half the distance of the code (but no further). Due to this "half the distance barrier", much effort has been devoted to designing codes with as large a distance as possible.

However, all the discussion above has not addressed one important aspect of decoding. We argued that for $\rho_U n = d/2 - 1$, there *exists* a unique transmitted codeword. However, the argument sheds no light on whether the decoder can find such a codeword *efficiently*. Of course, before we can formulate the question more precisely, we need to state what we mean by efficient decoding. We will formulate the notion more formally later on but for now we will say that a decoder is efficient if its running time is polynomial in the block length of the code (which is the number of symbols in the received word). As a warm up, let us consider the following naive decoding algorithm. The decoder goes through all the codewords in the code and outputs the codeword that is closest to the received word. The problem with this brute-force algorithm is that its running time is exponential in the block length for constant rate codes (which will be the focus of the first part of the thesis) and thus, is not an efficient algorithm. There is a rich body of beautiful work that focuses on designing efficient algorithms for unique decoding for many families of codes. These are discussed in detail in any standard coding theory texts such as [80, 104].

We now return to the gap between the half the distance and the information theoretic limit of $n - k$.

1.2.1 Going Beyond Half the Distance Bound

Let us revisit the bound of half the minimum distance on unique decoding. The bound follows from the fact that there exists an error pattern for which one cannot do unique decoding. However, such bad error patterns are rare. This follows from the nature of the space that the codewords (and the received words) "sit" in. In particular, one can think of a code of block length n as consisting of non-overlapping spheres of radius $d/2$, where the codewords are the centers of the spheres (see Figure 1.2). The argument for half the distance bound uses the fact that at least two such spheres touch. The touching point corresponds to the received word $\mathbf{y}$ that was used in the argument in the last section. However, the way the spheres pack in high dimension (recall the dimension of such a space is equal to the block length of the code n), almost every point in the ambient space has a unique by closest codeword at distances well beyond $d/2$ (see Figure 1.2).

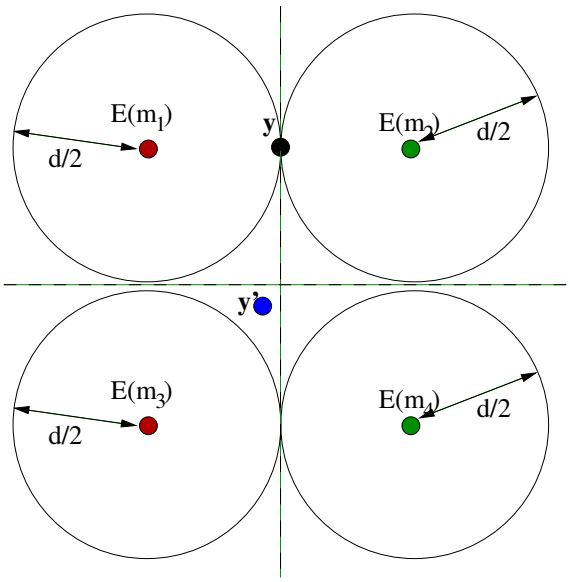

Figure 1.2: Four close by codewords $E(m_1), E(m_2), E(m_3)$ and $E(m_4)$ with two possible received words $\mathbf{y}$ and $\mathbf{y}'$. $E(m_1), E(m_2)$ and $\mathbf{y}$ form the bad example of Figure 1.1. However, the bad examples lie on the dotted lines. For example, $\mathbf{y}'$ is at a distance more than $d/2$ from its (unique) closest codewords $E(m_3)$. In high dimension, the space outside the balls of radius $d/2$ contains almost the entire ambient space.

Thus, by insisting on *always* getting back the original codeword, we are giving up on

correcting from error patterns from which we can recover the original codeword. One natural question one might ask is if one can somehow meaningfully relax this stringent constraint.

In the late 1950s, Elias and Wozencraft independently proposed a nice relaxation of unique decoding that gets around the barrier of half the distance bound [34, 106]. Under *list decoding*, the (list) decoder needs to output a "small" list of answers with the guarantee that the transmitted codeword is present in the list.[3] More formally, for a given error bound ρn and a received word $\mathbf{y}$, the list-decoding algorithm has to output all codewords that are at a distance at most ρn from $\mathbf{y}$. Note that when ρn is an upper bound on the number of errors that can be introduced by the channel, the list returned by the list-decoding algorithm will have the transmitted codeword in the list.

There are two immediate questions that arise: (i) Is list decoding a useful relaxation of unique decoding? (ii) Can we correct a number of errors that is close to the information theoretic limit using list decoding ?

Before we address these questions, let us first concentrate on a new parameter that this new definition throws into the mix: the worst case list size. Unless mentioned otherwise, we will use L to denote this parameter. Note that the running time of the decoding algorithm is $\Omega(L)$ as the decoder has to output every codeword in the list. Since we are interested in efficient, polynomial time, decoding algorithms, this puts an *a priori* requirement that L be a polynomial in the block length of the code. For a constant rate code, which has exponentially many codewords, the polynomial bound on L is very small compared to the total number of codewords. This bound was what we meant by small lists while defining list decoding.

Maximum Likelihood Decoding

We would like to point out that list decoding is not the only meaningful relaxation of unique decoding. Another relaxation called *maximum likelihood decoding* (or MLD) has been extensively studied in coding theory. Under MLD, the decoder must output the codeword that is closest to the received word. Note that if the number of errors is at most $(d-1)/2$, then MLD and unique decoding coincide. Thus, MLD is indeed a generalization of unique decoding.

MLD and list decoding are incomparable relaxations. On the one hand, if one can list decode efficiently up to the maximum number of errors that the channel can introduce then one can do efficient MLD. On the other hand, MLD does not put any restriction on the number of errors it needs to tolerate (whereas such a restriction is necessary for efficient list decoding). The main problem with MLD is that is turns out to be computationally intractable in general [17, 79, 4, 31, 37, 91] as well as for specific families of codes [66]. In

[3]The condition on the list size being small is important. Otherwise, here is a trivial list-decoding algorithm: output all codewords in the code. This, however is a very inefficient and more pertinently a useless algorithm. We will specify more carefully what we mean by small lists soon.

fact, there is no non-trivial family of codes known for which MLD can be done in polynomial time. However, list decoding is computationally tractable for many interesting families of codes (some of which we will see in this thesis).

We now turn to the questions that we raised about list decoding.

1.2.2 Why is List Decoding Any Good ?

We will now devote some time to address the question of whether list decoding is a meaningful relaxation of the unique decoding problem. Further, what does one do when the decoder outputs a list ?

In the communication setup, where the receiver might not have any side information, the receiver can still use a list-decoding algorithm to do "normal" decoding. It runs the list-decoding algorithm on the received word. If the list returned has just one codeword in it, then the receiver accepts that codeword as the transmitted codeword. If the list has more than one codeword, then it declares a decoding failure. First we note that this is no worse than the original unique decoding setup. Indeed if the number of errors is at most $d/2 - 1$, then by the discussion in Section 1.2 the list is going to contain one codeword and we would be back in the unique decoding regime. However, as was argued in the last section, for *most* error patterns (with total number of errors well beyond $d/2$) there is a unique closest by codeword. In other words, the list size for such error patterns would be one. Thus, list decoding allows us to correct from more error patterns than what was possible with unique decoding.

We now return to the question of whether list decoding can allow us to correct errors up to the information theoretic limit of $1 - R$? In short, the answer is yes. Using random coding arguments one can show that for any $\varepsilon > 0$, with high probability a random code of rate R, has the potential to correct up to $1 - R - \varepsilon$ fraction of errors with a worst case list size of $O(1/\varepsilon)$ (see Chapter 2 for more details). Further, one can show that for such codes, the list size is one for *most* received words.[4]

Other Applications of List Decoding

In addition to the immense practical potential of correcting more than half the distance number of errors in the communication setup, list decoding has found many surprising applications outside of the coding theory domain. The reader is referred to the survey by Sudan [98] and the thesis of Guruswami [49] (and the references therein) for more details on these applications. A key feature in all these applications is that there is some side information that one can use to sift through the list returned by the list-decoding algorithm to pick the "correct" codeword. A good analogy is that of a spell checker. Whenever a word is mis-spelt, the spell checker returns to the user a list of possible words that the user might have intended to use. The user can then prune this list to choose the word that he or she had

[4]This actually follows using the same arguments that Shannon used to establish his seminal result.

intended to use. Indeed, even in the communication setup, if the sender and the receiver can use a side channel (or have some shared information) then one can use list decoding to do "unambiguous" decoding [76].

1.2.3 The Challenge of List Decoding (and What Was Already Known)

In the last section, we argued that list decoding is a meaningful relaxation of unique decoding. More encouragingly, we mentioned that random codes have the potential to correct errors up to the information theoretic limit using list decoding. However, there are two major issues with the random codes result. First, these codes are not explicit. In real world applications, if one wants to communicate messages then one needs an explicit code. However, depending on the application, one might argue that doing a brute force search for such a code might work as this is a "one-off" cost that one has to pay. The second and perhaps more serious drawback is that the lack of structure in random codes implies that it is hard to come up with efficient list decodable algorithms for such codes. Note that for decoding, one cannot use a brute-force list-decoding algorithm.

Thus, the main challenge of list decoding is to come up with explicit codes along with efficient list-decoding (and encoding) algorithms that can correct errors close to the information theoretic limit of $n - k$.

The first non-trivial list-decoding algorithm is due to Sudan [97], which built on the results in [3]. Sudan devised a list-decoding algorithm for a specific family of codes called Reed-Solomon codes [90] (widely used in practice [105]), which could correct beyond half the distance barrier for Reed-Solomon codes of rate at most $1/3$. This result was then extended to work for all rates by Guruswami and Sudan [63]. It is worthwhile to note that even though list decoding was introduced in the late 1950s, these results came nearly forty years later. There was no improvement to the Guruswami-Sudan result until the recent work of Parvaresh and Vardy [85], who designed a code that is related to Reed-Solomon codes and presented efficient list-decoding algorithms that could correct more errors than the Guruswami-Sudan algorithm. However, the result of Parvaresh and Vardy does not meet the information theoretic limit (see Chapter 3 for more details). Further, for list decoding Reed-Solomon codes there has been no improvement over [63].

This concludes our discussion on the background for list decoding. We now turn to another relaxation of decoding that constitutes the second part of this thesis.

1.3 Property Testing of Error Correcting Codes

Consider the following communication scenario in which the channel is very noisy. The decoder, upon getting a very noisy received word, does its computation and ultimately reports a decoding failure. Typically, the decoding algorithm is an expensive procedure and it would be nice if one could quickly test if the received word is "far" from any codeword (in which case it should reject the received word) or is "close" to some codeword (in which case

it should accept the received word). In the former case, we would not run our expensive decoding algorithm and in the latter case, we would then proceed to run the decoding algorithm on the received word.

The notion of efficiency that we are going to consider for such spot checkers is going to be a bit different from that of decoding algorithms. We will require the spot checker to probe only a few positions in the received word during the course of its computation. Intuitively this should be possible as spot checking is a strictly easier task than decoding. Further, the fact that the spot checkers need to make their decision based on a portion of the received word should make spot checking very efficient. For example, if one could design spot checkers that look at only constant many positions (independent of the block length of the code), then we would have a spot checkers that run in constant time. However, note that since the spot checker cannot look at the whole received word it cannot possibly predict accurately if the received word is "far" from all the codewords or is "close" to some codeword. Thus, this notion of testing is a relaxation of the usual decoding as one sacrifices in the accuracy of the answer while gaining in terms of number of positions that one needs to probe.

A related notion of such spot checkers is that of locally testable codes (LTCs). LTCs have been the subject of much research over the years and there has been heightened activity and progress on them recently [46, 11, 74, 14, 13, 44]. LTCs are codes that have spot checkers as those discussed above with one crucial difference: they only need to differentiate between the cases when the received word is far from all codewords and the case when it *is* a codeword. LTCs arise in the construction of Probabilistically Checkable Proofs (PCPs) [5, 6] (see the survey by Goldreich [44] for more details on the interplay between LTCs and PCPs). Note that in the notion of LTC, there is no requirement on the spot checker for input strings that are very close to a codeword. This "asymmetry" in the way the spot checker accepts and rejects an input reflects the way PCPs are defined, where the emphasis is on rejecting "wrong" proofs.

Such spot checkers fall under the general purview of *property testing* (see for example the surveys by Ron [92] and Fischer [38]). In property testing, for some property P, given an object as an input, the spot checker has to decide if the given object satisfies the property P or is "far" from satisfying P. LTCs are a special case of property testing in which the property P is membership in some code and the objects are received words.

The ideal LTCs are codes with constant rate and linear distance that can be tested by probing only constant many position in the received word. However, unlike the situation in list decoding (where one can show the existence of codes with the "ideal" properties), it is *not known* if such LTCs exist.

1.3.1 A Brief History of Property Testing of Codes

The field of codeword testing, which started with the work of Blum, Luby and Rubinfeld [21] (who actually designed spot checkers for a variety of numerical problems), later developed into the broader field of property testing [93, 45]. LTCs were first explicitly de-

fined in [42, 93] and the systematic study of whether ideal LTCs (as discussed at the end of the last section) was initiated in [46]. Testing for Reed-Muller codes in particular has garnered a lot of attention [21, 9, 8, 36, 42, 93, 7, 1, 74], as they were crucial building blocks in the construction of PCPs [6, 5], Kaufman and Litsyn [73] gave a sufficient condition on an important class of codes that imply that the code is an LTC. Ben-Sasson and Sudan [13] built LTCs from a variant of PCPs called the Probabilistically Checkable Proof of Proximity– this "method" of constructing LTCs was initiated by Ben-Sasson et al. [11].

1.4 Contributions of This Thesis

The contributions of this thesis are in two parts. The first part deals with list decoding while the second part deals with property testing of codes.

1.4.1 List Decoding

This thesis advances our understanding of list decoding. Our results can be roughly divided into three parts: (i) List decodable codes of optimal rate over large alphabets, (ii) List decodable codes over small alphabets, and (iii) Limits to list decodability. We now look at each of these in more detail.

List Decodable Codes over Large Alphabets

Recall that for codes of rate R, it is information theoretically not possible to correct beyond $1-R$ fraction of errors. Further, using random coding argument one can show the existence of codes that can correct up to $1-R-\varepsilon$ fraction of errors for any $\varepsilon > 0$ (using list decoding). Since the first non-trivial algorithm of Sudan [97], there has been a lot of effort in designing explicit codes along with efficient list-decoding algorithms that can correct errors close to the information theoretic limit. In Chapter 3, we present the culmination of this line of work by presenting explicit codes (which are in turn extensions of Reed-Solomon codes) along with polynomial time list-decoding algorithm that can correct $1 - R - \varepsilon$ fraction of errors in polynomial time (for every rate $0 < R < 1$ and any $\varepsilon > 0$). This answers a question that has been open for close to 50 years and meets one of the central challenges in coding theory.

This work was done jointly with Venkatesan Guruswami and was published in the proceedings of the 38th Symposium on Theory of Computing (STOC), 2006 [58] and is under review for the journal IEEE Transactions on Information Theory.

List Decodable Codes over Small Alphabets

The codes mentioned in the last subsection are defined over alphabets whose size increases with the block length of the code. As discussed in Section 1.1.1, this is not a desirable feature. In Chapter 4, we show how to use our codes from Chapter 3 along with known

techniques of *code concatenation* and *expander graphs* to design codes over alphabets of size $2^{O(\varepsilon^{-4})}$ that can still correct up to $1 - R - \varepsilon$ fraction of errors for any $\varepsilon > 0$. To get to within ε of the information theoretic limit of $n - k$, it is known that one needs an alphabet of size $2^{\Omega(\varepsilon^{-1})}$ (see Chapter 2 for more details).

However, if one were interested in codes over alphabets of fixed size, the situation is different. First, it is known that for fixed size alphabets, the information theoretic limit is much smaller than $n - k$ (see Chapter 2 for more details). Again, one can show that random codes meet this limit. In Chapter 4, we present explicit codes along with efficient list-decoding algorithms that correct errors up to the so called Blokh-Zyablov bound. These results are the currently best known via explicit codes, though the number of errors that can be corrected is much smaller than the limit achievable by random codes.

This work was done jointly with Venkatesan Guruswami and appears in two different papers. The first was published in in the proceedings of the 38th Symposium on Theory of Computing (STOC), 2006 [58] and is under review for the journal IEEE Transactions on Information Theory. The second paper will appear in the proceedings of the 11th International Workshop on Randomization and Computation (RANDOM) [60].

Explicit codes over fixed alphabets, considered in Chapter 4, are constructed using *code concatenation*. However, as mentioned earlier, the fraction of errors that such codes can tolerate via list decoding is far from the information theoretic limit. A natural question to ask is whether one can use concatenated codes to achieve the information theoretic limit? In Chapter 5 we give a positive answer to this question in following sense. We present a random ensemble of concatenated codes that with high probability meet the information theoretic limit: That is, they can potentially list decode as large a fraction of errors as general random codes, though with larger lists.

This work was done jointly with Venkatesan Guruswami and is an unpublished manuscript [61].

Limits to List Decoding Reed-Solomon Codes

The results discussed in the previous two subsections are of the following flavor. We know that random codes allow us to list decode up to a certain number of errors, and that is optimal. Can we design more explicit codes (maybe with efficient list-decoding algorithms) that can correct close to the number of errors that can be corrected by random codes? However, consider the scenario where one is constrained to work with a certain family of codes, say Reed-Solomon codes. Under this restriction what is the most number of errors from which one can hope to list decode?

The result of Guruswami and Sudan [63] says that one can efficiently correct up to $n - \sqrt{nk}$ many errors for Reed-Solomon codes. However, is this the best possible? In Chapter 6, we give some evidence that the Guruswami-Sudan algorithm might indeed be the best possible. Along the way we also give some explicit constructions of "bad list-decoding configurations." A bad list-decoding configuration refers to a received word $\mathbf{y}$ along with an

error bound ρ such that there are super-polynomial (in n) many Reed-Solomon codewords within a distance of ρn from $\mathbf{y}$.

This work was done jointly with Venkatesan Guruswami and appears in two different papers. The first was published in in the proceedings of the 37th Symposium on Theory of Computing (STOC), 2005 [56] as well as in the IEEE Transactions on Information Theory [59]. The second paper is an unpublished manuscript [62].

1.4.2 Property Testing

We now discuss our results on property testing of error correcting codes.

Testing Reed-Muller Codes

Reed-Muller codes are generalizations of Reed-Solomon codes. Reed-Muller codes are based on multivariate polynomials while Reed-Solomon codes are based on univariate polynomials. Local testing of Reed-Muller codes was instrumental in many constructions of PCPs. However, the testers were only designed for Reed-Muller codes over large alphabets. In fact, the size of the alphabet of such codes depends on the block length of the codes. In Chapter 7, we present near-optimal local testers for Reed-Muller codes defined over (a class of) alphabets of fixed size.

This work was done jointly with Charanjit Jutla, Anindya Patthak and David Zuckerman and was published in the proceedings of the 45th Symposium on Foundation of Computer Science (FOCS), 2005 [72] and is currently under review for the journal Random Structures and Algorithms.

Tolerant Locally Testable Codes

Recall that the notion of spot checkers that we were interested in had to accept the received word if it is far from all codewords and reject when it is close to some codeword (as opposed to LTCs, which only require to accept when the received word is a codeword). Surprisingly, such testers were not considered in literature before. In Chapter 8, we define such testers, which we call tolerant testers. Our results show that in general LTCs do not imply tolerant testability, though most LTCs that achieve the best parameters also have tolerant testers.

As a slight aside, we look at certain strong form of local testability (called *robust testability*) of certain *product of codes*. Product of codes are also special cases of certain concatenated codes considered in Chapter 4. We show that in general, certain product of codes cannot be robustly testable.

This work on tolerant testing was done jointly with Venkatesan Guruswami and was published in the proceedings of the 9th International Workshop on Randomization and Computation (RANDOM) [57]. The work on robust testability of product of codes is joint work with Don Coppersmith and is an unpublished manuscript [26].

1.4.3 Organization of the Thesis

We start with some preliminaries in Chapter 2. In Chapter 3, we present the main result of this thesis: codes with optimal rate over large alphabets. This result is then used to design new codes in Chapters 4 and 5. We present codes over small alphabets in Chapter 4, which are constructed by a combination of the codes from Chapter 3 and *code concatenation*. In Chapter 5, we show that certain random codes constructed by code concatenation also achieve the list-decoding capacity. In Chapter 6, we present some limitations to list decoding Reed-Solomon codes. We switch gears in Chapter 7 and present new local testers for Reed-Muller codes. We present our results on tolerant testability in Chapter 8. We conclude with the major open questions in Chapter 9.